

A CEGAR-like Approach for Cost LTL Bounds

Maximilien Colange¹, Dimitri Racordon¹, and Didier Buchs¹

¹Centre Universitaire d'Informatique – University of Geneva, first.last@unige.ch

Abstract

Qualitative formal verification, that seeks boolean answers about the behavior of a system, is often insufficient for practical purposes. Observing quantitative information is of interest, e.g. for the proper calibration of a battery or a real-time scheduler. Historically, the focus has been on quantities in a continuous domain, but recent years showed a renewed interest for discrete quantitative domains.

Cost Linear Temporal Logic (CLTL) is a quantitative extension of classical LTL. It integrates into a nice theory developed in the past few years that extends the qualitative setting, with counterparts in terms of logics, automata and algebraic structure. We propose a practical usage of this logics for model-checking purposes. A CLTL formula defines a function from infinite words to integers. Finding the bounds of such a function over a given set of words can be seen as an extension of LTL universal and existential model-checking. We propose a CEGAR-like algorithm to find these bounds by relying on classical LTL model-checking, and use Büchi automata with counters to implement it. This method constitutes a first step towards the practical use of such a discrete quantitative logic.

1 Introduction

Qualitative verification, asking questions with boolean answers about a system may be too strict for various applications. Calibrating a battery, timing a scheduler, measuring quality of service are practical problems of systems designers for which formal verification can offer a guarantee. Many works focus on the case of continuous quantitative domains (stochastic systems, real-time systems ...), and the case of discrete domains have long been overlooked.

The ability to count events is an important feature, e.g. to evaluate logical time (number of actions done by a robot, number of context switches done by a scheduler ...). Such measurements are of primary interest to evaluate the behavior of a system at early stages of development. Logical time can also serve as a first approximation of real-time, when events have a known bounded duration. We seek in this paper to use a logic able to count events in a system with infinite behaviors, with a focus on applicability. Following the automata-approach largely adopted for Linear Temporal Logic (LTL) verification, we study a quantitative extension of automata able to count events.

Among numerous quantitative extensions to finite automata, we focus on the one defined by Colcombet and Bojańczyk [5]. Finite automata are extended with a finite set of counters that can be incremented and reset. A special operation

observe allows to store the current value of a counter to further determine the value of a run, as the infimum or the supremum of such stored values. They are part of a vast theory that nicely extends the finite automata theory, with their logical and algebraic counterparts, closure properties, over finite and infinite words and finite trees. Such automata define functions from words to integers, termed *cost functions*. Due to the undecidability of comparing two cost functions, many nice features of this theory rely on the consideration of cost functions up to an equivalence relation that erases the exact values and only retains their boundedness of functions.

Regarding infinite words, on which we focus in this paper, the theory of cost functions has links with other extensions of automata or logics, by bounding a discrete quantity: bounding the maximal time between two returns to an accepting state in ω -automata [2], or bounding the wait time for the *finally* operator in LTL [17]. Considering exact values to count events is nevertheless a great tool for verification. Think for example to the maximal number of energy units consumed by a robot between two returns to its charging base, to calibrate a battery. Or the maximal number of simultaneous threads in a parallel computation, to tune an appropriate scheduler. Or the number of false steps permitted to a human operator before a safeguard restriction occurs. For such properties, determining whether the bound is finite or not is of little help. We thus propose to use the tools and methods developed towards cost function theory (over infinite words) to practical model-checking.

We use a counting extension of LTL, Cost LTL, introduced in [16] in the context of the theory of cost functions. Our contribution is threefold:

- Cost LTL being an extension of LTL, we show how classical model-checking problems on LTL extend to the quantitative case. We thus address the problem of finding the bounds of the cost function defined by a formula, with a focus on the upper bound search. We propose an algorithm to compute such an upper bound using a CEGAR-like approach, where the bound is computed thanks to successive refinements.
- We show how this algorithm is effective, implementing it by means of ω -automata with counters. The computational bottleneck of the algorithm is reduced to Büchi automata emptiness check, to take advantage of existing research in the field.
- We also present concrete examples of application of Cost LTL, to illustrate its potential as a practical tool for verification.

The paper is organized as follows: Section 2 first presents Cost LTL, introduced in [16], and some basic results used in the remainder of the paper. The core of our contribution is a CEGAR-like algorithm to determine the bounds of a cost function defined by a CLTL property, in Section 3. We then show in Section 4 how this algorithm can be effectively implemented thanks to ω -automata equipped with counters. Finally, Section 6 presents related work, and Section 7 concludes our study and proposes leads for future developments.

Notations Given $u \in \Sigma^\omega$ an infinite word over an alphabet Σ , and $i \in \mathbb{N}$, u_i is the i -th letter of u , and u^i the suffix of u starting at u_i . Thus $u = u^0$ and $u = u_0 \dots u_{i-1} u^i$ for any $i > 0$. If A is a finite set, $|A|$ denotes its cardinal. For $A \subseteq \mathbb{N}$, $\inf A$ (resp. $\sup A$) denotes the infimum (resp. supremum) of A . By convention, $\inf \emptyset = +\infty$ and $\sup \emptyset = 0$. Let a $f \in \mathbb{N}^D$ for some set D . For

$D' \subseteq D$, the image of D' by f is the set $f(D') = \{f(x) \mid x \in D'\}$. We also note $\sup_{D'} f = \sup f(D')$, respectively $\inf_{D'} f = \inf f(D')$.

2 Cost Linear Temporal Logics

We first define Cost Linear Temporal Logic (LTL^{\leq}), as in [16]. Let AP be a set of atomic propositions. The set of LTL^{\leq} formulae is defined by (a ranges over AP):

$$\phi ::= a \mid \neg a \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi \mathbf{U} \phi \mid \phi \mathbf{R} \phi \mid \mathbf{X} \phi \mid \phi \mathbf{U}^{\leq} \phi$$

Every LTL formula has a semantically equivalent formula in Negative Normal Form (NNF), where negations can only appear in front of an atomic proposition. Any LTL formula in NNF is a LTL^{\leq} formula, and in that sense, LTL is a strict subset of LTL^{\leq} . From now on, we identify LTL with LTL in NNF, so that $\text{LTL} = \text{LTL}^{\leq} - \{\mathbf{U}^{\leq}\}$ and $\text{LTL} \subsetneq \text{LTL}^{\leq}$.

A formula of LTL^{\leq} is evaluated over infinite words on the alphabet 2^{AP} . Let $u \in (2^{AP})^{\omega}$, $n \in \mathbb{N}$, ϕ_1, ϕ_2 be LTL^{\leq} formulae, and $a \in AP$:

$$\begin{aligned} (u, n) \models_{\leq} a & \quad \text{iff } a \in u_n \\ (u, n) \models_{\leq} \neg a & \quad \text{iff } a \notin u_n \\ (u, n) \models_{\leq} \phi_1 \vee \phi_2 & \quad \text{iff } (u, n) \models_{\leq} \phi_1 \text{ or } (u, n) \models_{\leq} \phi_2 \\ (u, n) \models_{\leq} \phi_1 \wedge \phi_2 & \quad \text{iff } (u, n) \models_{\leq} \phi_1 \text{ and } (u, n) \models_{\leq} \phi_2 \\ (u, n) \models_{\leq} \mathbf{X} \phi_1 & \quad \text{iff } (u^1, n) \models_{\leq} \phi_1 \\ (u, n) \models_{\leq} \phi_1 \mathbf{U} \phi_2 & \quad \text{iff } \exists i \in \mathbb{N} \text{ s.t. } (u^i, n) \models_{\leq} \phi_2 \\ & \quad \text{and } \forall j < i, (u^j, n) \models_{\leq} \phi_1 \\ (u, n) \models_{\leq} \phi_1 \mathbf{R} \phi_2 & \quad \text{iff } \forall i \in \mathbb{N} \text{ either } (u^i, n) \models_{\leq} \phi_2 \\ & \quad \text{or } \exists j < i \text{ s.t. } (u^j, n) \models_{\leq} \phi_1 \\ (u, n) \models_{\leq} \phi_1 \mathbf{U}^{\leq} \phi_2 & \quad \text{iff } \exists i \in \mathbb{N} \text{ s.t. } (u^i, n) \models_{\leq} \phi_2 \\ & \quad \text{and } |\{j < i \mid (u^j, n) \not\models_{\leq} \phi_1\}| \leq n \end{aligned}$$

The semantics of $\phi \in \text{LTL}^{\leq}$ is the function

$$\llbracket \phi \rrbracket_{\leq} : \begin{array}{ll} (2^{AP})^{\omega} & \mapsto \mathbb{N} \cup \{\infty\} \\ u & \mapsto \inf \{n \mid (u, n) \models_{\leq} \phi\} \end{array}$$

To keep examples clear, we identify any atomic proposition a with the subset of 2^{AP} of sets containing a , \bar{a} being its complementary. Consider the formula $\phi_1 = \mathbf{F}^{\leq} \neg a$, short for $\perp \mathbf{U}^{\leq} \neg a$. For any $n \leq p$ and any word $u \in a^n \bar{a} (2^{AP})^{\omega}$, $(u, p) \models_{\leq} \phi_1$, and $\llbracket \phi_1 \rrbracket_{\leq}(u) = n$. Consider now the formula $\phi_2 = \mathbf{G}(\mathbf{F}^{\leq} \neg a)$, short for $\perp \mathbf{R} \mathbf{F}^{\leq} \neg a$. For $n \in \mathbb{N}$ and $u \in (2^{AP})^{\omega}$, $(u, n) \models_{\leq} \phi_2$ only if the distance between a letter in \bar{a} and the next one also in \bar{a} never exceeds n . Thus $\llbracket \phi_2 \rrbracket_{\leq}(u)$ is the maximal number of consecutive a 's in u .

If ϕ is a LTL formula, either $(u, n) \models_{\leq} \phi$ holds for every n , in which case $\llbracket \phi \rrbracket_{\leq}(u) = 0$, or for none, in which case $\llbracket \phi \rrbracket_{\leq}(u) = \infty$. The former is noted $u \vdash \phi$, and matches the usual semantics of LTL. In other words, the value **true** is mapped onto 0 and **false** onto ∞ .

From the semantical definition, for any integers $n \leq p$, if $(u, n) \models_{\leq} \phi$ then $(u, p) \models_{\leq} \phi$ too. Stating $(u, n) \models_{\leq} \phi$ is thus equivalent to stating $\llbracket \phi \rrbracket_{\leq}(u) \leq n$.

For $n \in \mathbb{N}$, we propose a translation from a LTL^{\leq} formula ϕ to a LTL formula $\phi[n]$ that separates words according to their value relatively to n . More precisely, $u \vdash \phi[n]$ if, and only if, $\llbracket \phi \rrbracket_{\leq}(u) \leq n$. $\phi[n]$ is defined inductively as follows ($\phi_1, \phi_2 \in LTL^{\leq}$ and $a \in AP$):

- $a[n] = a$ and $(\neg a)[n] = \neg a$;
- $(\mathbf{X} \phi_1)[n] = \mathbf{X}(\phi_1[n])$
- $(\phi_1 \bowtie \phi_2)[n] = \phi_1[n] \bowtie \phi_2[n]$ for $\bowtie \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}$;
- for $\phi_1, \phi_2 \in LTL$, $(\phi_1 \mathbf{U}^{\leq} \phi_2)[0] = \phi_1 \mathbf{U} \phi_2$
and $(\phi_1 \mathbf{U}^{\leq} \phi_2)[n+1] = (\phi_1 \vee \mathbf{X}(\phi_1 \mathbf{U}^{\leq} \phi_2)[n]) \mathbf{U} \phi_2$;
- otherwise $(\phi_1 \mathbf{U}^{\leq} \phi_2)[n] = (\phi_1[n] \mathbf{U}^{\leq} \phi_2[n])[n]$.

Back to our example $\phi_1 = \mathbf{F}^{\leq} \neg a$, we have $\phi_1[0] = \perp \mathbf{U} \neg a = \neg a$, hence $\phi_1[1] = (\mathbf{X} \neg a) \mathbf{U} \neg a$, equivalent to $\neg a \vee \mathbf{X} \neg a$. Thus, $\phi_1[n] = \bigvee_{i=0}^n X^i \neg a$ for every n .

Property 1. For $u \in (2^{AP})^\omega$, $n \in \mathbb{N}$ and $\phi \in LTL^{\leq}$, $u \vdash \phi[n]$ iff $\llbracket \phi \rrbracket_{\leq}(u) \leq n$.

Proof. Structural induction on ϕ , detailed in appendix. \square

2.1 Dual Logics

In LTL^{\leq} , negations can only appear in the leaves of the formula, so that a formula is always in NNF. This particularity is commanded by the semantical difficulty to negate the operator \mathbf{U}^{\leq} . In a boolean setting, a word is either a model of the formula, or it is not. In our quantitative setting, negation is not straightforward, as it is not a natural operation over \mathbb{N} . We take inspiration from the embedment of LTL in LTL^{\leq} : **true** corresponds to 0 and **false** to ∞ . Semantically, the negation thus replaces inf with sup. We define the logic $LTL^>$ [15], dual to LTL^{\leq} . The operator \mathbf{U}^{\leq} is replaced by $\mathbf{R}^>$ whose semantics is defined so as to match the negation of \mathbf{U}^{\leq} semantics: $(u, n) \models_{>} \phi_1 \mathbf{R}^> \phi_2$ iff for every $i \in \mathbb{N}$, either $(u^i, n) \models_{>} \phi_2$ or $|\{j < i \mid (u^j, n) \models_{>} \phi_1\}| > n$. All other operators keep their natural semantics. The semantics of $\phi \in LTL^>$ is a function

$$\llbracket \phi \rrbracket_{>} : \begin{array}{ll} (2^{AP})^\omega & \mapsto \mathbb{N} \cup \{\infty\} \\ u & \mapsto \sup \{n \mid (u, n) \models_{>} \phi\} \end{array}$$

For any $n \leq p$, if $(u, p) \models_{>} \phi$ then $(u, n) \models_{>} \phi$, so that $(u, n) \models_{>} \phi$ iff $\llbracket \phi \rrbracket_{>}(u) \geq n$.

Note that LTL is also embedded in $LTL^>$, with a semantics dual to the case of LTL^{\leq} : **true** is mapped onto ∞ and **false** onto 0. Syntactically, $LTL = LTL^{\leq} \cap LTL^>$, but the semantics do not match. Note that in both cases (LTL^{\leq} and $LTL^>$), either (u, n) is a model for $\phi \in LTL$ for every $n \in \mathbb{N}$, or for none. From now on, we note $u \vdash \phi$ for the former case, to be matched with the appropriate semantics depending on context.

Syntactically, we get dual pairs of operators: \vee and \wedge , \mathbf{U} and \mathbf{R} , \mathbf{U}^{\leq} and $\mathbf{R}^>$, \mathbf{X} being self-dual. From a formula $\phi \in LTL^{\leq}$ (resp. $LTL^>$), we can build a formula $\neg \phi \in LTL^>$ (resp. LTL^{\leq}), by pushing the negation to the leaves: the top operator is replaced by its dual, and the negation is recursively pushed to the leaves of its operands. Literals respect the excluded middle, which is semantically consistent, so as to eliminate double negations. Syntactically, the excluded middle also holds, as pushing negations to the leaves in $\neg \neg \phi$ yields ϕ . Observe how the semantics of ϕ and $\neg \phi$ are correlated:

Property 2. For $u \in (2^{AP})^\omega$ and $\phi \in LTL^\leq$, $\llbracket \neg\phi \rrbracket_>(u) = \max(0, \llbracket \phi \rrbracket_\leq(u) - 1)$.

Proof. An easy induction on ϕ proves that $(u, n) \models_\leq \phi$ iff $(u, n) \not\models_> \neg\phi$ (observe how the semantics of dual operators are dual to each other). We recall that $(u, n) \models_\leq \phi \implies (u, p) \models_\leq \phi$ for any $n \leq p$. Thus $(u, n) \models_> \neg\phi$ if, and only if, $n < \llbracket \phi \rrbracket_\leq(u) = \inf \{p \mid (u, p) \models_\leq \phi\}$. Furthermore, if $\llbracket \phi \rrbracket_\leq(u) = 0$, then $\llbracket \phi \rrbracket_>(u) = \sup \emptyset = 0$ by convention. \square

Property 3. For any $u \in (2^{AP})^\omega$ and $\phi \in LTL^>$,

$$\llbracket \neg\phi \rrbracket_\leq(u) = \begin{cases} \llbracket \phi \rrbracket_>(u) + 1 & \text{if } \llbracket \phi \rrbracket_>(u) > 0 \\ 0 & \text{if } \forall n \in \mathbb{N}. (u, n) \not\models_> \phi \\ 1 & \text{otherwise} \end{cases}$$

Proof. Recall from above that $(u, n) \models_> \phi$ if, and only if, $(u, n) \not\models_> \neg\phi$. If $\llbracket \phi \rrbracket_>(u) > 0$, then $(u, n) \models_\leq \phi$ if, and only if, $n > \llbracket \phi \rrbracket_>(u)$. If $\llbracket \phi \rrbracket_>(u) = 0$, $\{n \mid (u, n) \models_> \phi\}$ is either \emptyset or $\{0\}$. We get $\llbracket \neg\phi \rrbracket_\leq(u) = 0$ in the former case, and $\llbracket \neg\phi \rrbracket_\leq(u) = 1$ in the latter. \square

Following the above, we define, for $\phi \in LTL^>$ and $n \in \mathbb{N}$, $\phi[n]$ as $\neg((\neg\phi)[n])$.

Property 4. For $u \in (2^{AP})^\omega$, $n \in \mathbb{N}$ and $\phi \in LTL^>$, $u \vdash \phi[n]$ iff $\llbracket \phi \rrbracket_>(u) \geq n$.

Proof. $\llbracket \phi \rrbracket_>(u) \geq n$ iff $(u, n) \models_> \phi$ iff $(u, n) \not\models_\leq \neg\phi$ iff $u \not\vdash (\neg\phi)[n]$ iff $u \vdash \phi[n]$. \square

2.2 Cost Logics for Verification

A common task in verification is whether a system has a behavior satisfying a given property. The property either expresses a desired behavior, or an unwanted one (in which case finding a satisfying behavior amounts to finding a bug). Typically, the behaviors of the system is a regular ω -language L , and the property ϕ is a LTL property. Thanks to the closure properties of regular ω -languages, this problem reduces to *existential model-checking*: is the intersection of L and the language recognized by ϕ empty? *Universal model-checking* asks whether a language L' contains all ω -words, and is dual to existential model-checking, since $L' = \Sigma^\omega$ iff $\Sigma^\omega - L' = \emptyset$. As both LTL^\leq and $LTL^>$ extend LTL, the natural question we address is the extension of these two problems to the quantitative setting.

We first rephrase the LTL existential and universal model-checking with the LTL^\leq semantics: existential model-checking asks whether there is a word of value 0. Dually, universal model-checking asks whether all words have value ∞ . Existence of a word of value equal to, greater than, or less than a given n are natural extensions of this question. These questions hardly extend the boolean framework: by comparing word values against a *given* n , they remain boolean questions. We seek here a question with a *quantitative* answer (in our case in the domain $\mathbb{N} \cup \{\infty\}$). Two particular values of interest are the bounds of $\llbracket \phi \rrbracket_\leq$.

Definition 5. *inf-bound checking*: given $L \subseteq (2^{AP})^\omega$ and $\phi \in LTL^\leq$, compute $\inf_L \llbracket \phi \rrbracket_\leq$.

Definition 6. *sup-bound checking*: given $L \subseteq (2^{AP})^\omega$ and $\phi \in LTL^\leq$, compute $\sup_L \llbracket \phi \rrbracket_\leq$.

The duality of LTL^{\leq} and $LTL^{>}$ allows to choose from ϕ or $\neg\phi$, as $LTL^{>}$ seems more appropriate for sup-bound checking. All the problems mentioned above are reducible to these two problems. In LTL^{\leq} semantics, existential LTL model-checking boils down to inf-bound checking, while its universal counterpart corresponds to sup-bound checking.

3 CLTL Bounds Checking

This section presents our main contribution, an algorithm to compute bounds for LTL^{\leq} and $LTL^{>}$ formulae. It is inspired by the Counter-Examples Guided Abstraction and Refinement (CEGAR) approach to qualitative model-checking, which we present first.

3.1 The CEGAR Approach to qualitative model-checking

Consider a regular ω -language L (the set of behaviors of a system) and a LTL formula ϕ . We ask whether all words of L are models of ϕ . It boils down to existential model-checking: is there a model of $\neg\phi$ in L ? The language of $\neg\phi$, $\mathcal{L}(\neg\phi)$ is in deed regular, the intersection of $\mathcal{L}(\neg\phi)$ and L computable, and testing the emptiness of a regular ω -language is decidable. These steps are usually performed with ω -automata to represent the regular ω -languages.

But this approach becomes hardly tractable when the underlying automata are huge, as it is often the case when the input language is the set of behaviors of a concurrent system. We present the so-called CEGAR (Counter-Example Guided Abstraction and Refinement) loop. A language L' larger (for the inclusion) than L is called an *abstraction*, L being a *refinement* of L' . CEGAR loop assumes the existence of a refinement function ρ that given a word $u \notin L$ and a regular abstraction M of L , returns a regular refinement M' of M that does not contain u , and that is also an abstraction of L : $L \subseteq M$ and $u \notin L \Rightarrow L \subseteq \rho(M, u) \subseteq M - \{u\}$.

The CEGAR loop proceeds as follows: i) start from an abstraction M of L ; ii) search in M a model u for $\neg\phi$; iii) if there is no such u in M , there is none in L either, and the question is settled; iv) otherwise, check whether $u \in L$; v) if $u \in L$, then the question is settled; vi) else start over with $M = \rho(M, u)$. In practice, the automaton for L is huge, and CEGAR avoids its full exploration by manipulating abstractions, that have smaller underlying automata. Since a counter-example guides the refinement, the same spurious counter-example cannot be encountered twice. In general, termination depends on the refinement function ρ , but practically, termination is easy to ensure, for example by falling back to the initial input L (worst-case scenario) when the size of M' exceeds the size of L .

3.2 CEGAR Approach for Bounds Checking

We adapt the CEGAR approach to solve sup-bound checking for a $LTL^{>}$ formula ϕ over a regular ω -language L . The dual inf-bound checking follows the same scheme.

We first have to define the notion of abstractions and refinements, thanks to an ordering over semantic functions: smaller elements are refinements and

greater ones are abstractions.

Definition 7. For $L \subseteq \Sigma^\omega$, $f \preceq_L g$ iff $\sup_L f = \sup_L g$ and $g^{-1}(0) \subseteq f^{-1}(0)$.

Algorithm 1: ComputeBound(L, ϕ_0)

```

1  $n \leftarrow 0$  ;
2 while true do
3    $\phi \leftarrow \phi_0 \wedge \phi_0[n+1]$  ;
4   if  $\exists u \in L$  s.t.  $\llbracket \phi \rrbracket_>(u) > 0$ 
     then
5      $n \leftarrow p$  for any
        $n < p \leq \llbracket \phi \rrbracket_>(u)$  ;
6   else
7     return  $n$ ;
```

In other words, f refines g (relatively to L) if they have the same supremum over L and f maps more words (for the inclusion) onto 0 than g . The CEGAR loop for our quantitative setting is shown in Algorithm 1.

We present the algorithm in its full generality. Line 5 leaves some room for various implementations, as we will see in Section 4.2.

The key of this algorithm is the search for a word u such that $\llbracket \phi \rrbracket_>(u) > 0$ for a LTL[>] formula ϕ (line 4). Note that if $\sup_L \llbracket \phi \rrbracket_> > 0$, then for all $u \in L$, $\llbracket \phi \rrbracket_>(u) > 0$ iff there is some p such that $(u, p) \models_> \phi$. Considering the semantics of LTL[>], this is equivalent to finding a word satisfying the LTL formula ϕ' , a copy of ϕ in which every occurrence of $\phi_1 \mathbf{R}^> \phi_2$ is replaced by $\top \mathbf{R} \phi_2$ (where $\top = a \vee \neg a$ for any $a \in AP$). Therefore, the search for the upper bound of $\llbracket \phi \rrbracket_>$ is reduced to LTL emptiness check, a well-studied problem with numerous efficient solutions (see [22, 23] for surveys). The corner case $\sup_L \llbracket \phi \rrbracket_> = 0$ can be detected at the second pass in the loop (see the proof of Proposition 9).

Property 8. At line 4, for all $u \in L$, $\llbracket \phi \rrbracket_>(u) = \begin{cases} \llbracket \phi_0 \rrbracket_>(u) & \text{if } \llbracket \phi_0 \rrbracket_>(u) > n \\ 0 & \text{otherwise} \end{cases}$

Proof. At line 4, $\phi = \phi_0 \wedge \phi_0[n+1]$. Let $u \in L$ and $m = \llbracket \phi_0 \rrbracket_>(u)$. By Proposition 4, $m > n$ iff $(u, m) \models_> \phi_0[n+1]$. $\llbracket \phi \rrbracket_>(u)$ is the largest p such that both $(u, p) \models_> \phi_0$ and $(u, p) \models_> \phi_0[n+1]$. If $m > n$, m is the largest such p , so that $\llbracket \phi \rrbracket_>(u) = m$. Otherwise, there are no value complying to the latter condition, and $\llbracket \phi \rrbracket_>(u) = \sup \emptyset = 0$. \square

Proposition 8 proves that $\llbracket \phi \rrbracket_> \preceq_L \llbracket \phi_0 \rrbracket_>$, and that at each pass in the loop, ϕ is refined with respect to \preceq_L .

Property 9. If $\sup_L \llbracket \phi_0 \rrbracket_>$ is finite, **ComputeBound** is both correct and sound, i.e. it terminates and returns $\sup_L \llbracket \phi_0 \rrbracket_>$.

Proof. At line 5, n is updated with a value p such that $n < p \leq \llbracket \phi_0 \rrbracket_>(u)$ for some $u \in L$ such that $\llbracket \phi \rrbracket_>(u) > 0$. Proposition 8 guarantees the existence of such a p , and n strictly increases when updated. n is obviously bounded by $\sup_L \llbracket \phi_0 \rrbracket_>$, which proves termination. Moreover, as long as $n < \sup_L \llbracket \phi_0 \rrbracket_>$, there are still words u such that $\llbracket \phi_0 \rrbracket_>(u) > n$, i.e. $\llbracket \phi \rrbracket_>(u) > 0$. If the search for such words on line 4 is correct and sound, so is **ComputeBound**. \square

3.3 Performance of the algorithm

Essentially, **ComputeBound** enumerates candidate values for $\sup \llbracket \phi_0 \rrbracket_>$ in increasing order until a fixpoint is reached. The next candidate is determined on line 5:

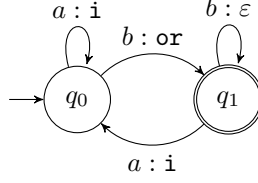


Figure 1: A CA counting consecutive a 's

the larger p , the quicker the algorithm converges. The choice left for p allows flexibility: the exact value of $\llbracket \phi \rrbracket_{>}(u)$ is most certainly harder to find than an appropriate value p . This line brings a tuning parameter for implementations: the higher the p , the faster the convergence, but probably the higher the computation cost. Implementations should therefore find an appropriate balance between the cost of computing p , and the number of loops in **ComputeBound**.

To a lesser extent, line 3 brings another tuning parameter for implementations. ϕ_0 (resp. $\phi_0[n+1]$) in this line can be safely replaced by ϕ (resp. $\phi[n+1]$), without affecting the outcome of the algorithm. Nevertheless, using the ϕ_0 variant yields simpler formulae.

4 Counter ω -Automata

This section presents Counter ω -Automata (CA), as introduced by [5] under the names B -automata and S -automata. We also show how to translate $LTL^>$ formulae to CA, based on ideas used for the case of finite words [15]. We adapt it to infinite words in Section 4.1, and then show the implementation of our **ComputeBound** algorithm with CA in Section 4.2.

Informally, a CA is a ω -automaton equipped with a finite set of non-negative integer counters Γ , initialized with value 0. The values of these counters are controlled by actions: **i** that increments a counter; **r** that resets a counter to 0; **o** that *observes*, or stores, the current value of the counter. The set of counter actions is denoted by \mathbb{C} . Values of the counters do not affect the behavior of the automaton, but are used to assign a *value* to a word. Only observed values are used to determine word values. In addition to a letter $a \in \Sigma$, a CA transition is labelled with $|\Gamma|$ (words of) actions, one for each counter.

Definition 10. A counter automaton is a 6-tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, \Gamma, q_0, \mathcal{F} \rangle$ where:

- Q is a finite set of states, and $q_0 \in Q$ is the initial state;
- Σ is a finite alphabet;
- Γ is a finite set of counters;
- $\Delta \subseteq Q \times \Sigma \times (\mathbb{C}^*)^\Gamma \times Q$ is the transition relation;
- $\mathcal{F} \subseteq 2^\Delta$ is a set of sets of accepting transitions.

An infinite word $u \in \Sigma^\omega$ is accepted by a Counter Automaton \mathcal{A} if there exists an execution of \mathcal{A} on u that visits infinitely often every set in \mathcal{F} .

For $\Gamma = \emptyset$, Definition 10 defines a ω -automaton. Along a run ρ , counters are incremented and reset according to the encountered actions, and the set of checked values is noted $C(\rho)$.

There are two dual semantics for CA:

- the inf-semantics (B -automata in [5]), where
 $\llbracket \mathcal{A} \rrbracket_{\leq}(u) = \inf_{\rho \text{ acc. run on } u} \sup C(\rho)$;
- the sup-semantics (S -automata in [5]), where
 $\llbracket \mathcal{A} \rrbracket_{>}(u) = \sup_{\rho \text{ acc. run on } u} \inf C(\rho)$.

Figure 1 gives an example of a deterministic CA with the sup-semantics. Only words in $L = (\Sigma^*b)^\omega$ have accepting runs. Thus $\llbracket \mathcal{A} \rrbracket_{>}(u) = \sup \emptyset = 0$ for $u \notin L$. If $u \in L$, $\llbracket \mathcal{A} \rrbracket_{>}(u)$ is the smallest size of a block of consecutive a 's in u .

4.1 From CLTL to Counter Automata

For every LTL ^{\leq} (resp. LTL ^{$>$}) formula ϕ , there exists a CA \mathcal{A}_ϕ with the inf-semantics (resp. sup-semantics) with the same semantics: $\llbracket \mathcal{A} \rrbracket_{\leq} = \llbracket \phi \rrbracket_{\leq}$ (resp. $\llbracket \mathcal{A} \rrbracket_{>} = \llbracket \phi \rrbracket_{>}$). This construction is effective, and does not differ much from the translation from LTL formulae to Büchi automata (see for instance [8]). The key difference is the introduction of a counter for each occurrence of the operator \mathbf{U}^{\leq} (resp. $\mathbf{R}^>$) in the formula to translate. The translation is described in [15] for the case of finite words, and is easily extended to infinite words.

We state here this extension, for the sake of completeness. In [15], the produced CA transitions bear *sequences* of counter actions (e.g. a counter can be incremented by three in a single transition). We show that it is always possible to produce a CA whose transitions are labelled with *atomic actions*, i.e. at most one action (\mathbf{i} , \mathbf{or} or ε) per counter. This possibility seems to have been overlooked in previous work. This remark may stem from the care taken in our translation to retain exact values. We also note that there is a slight difference of semantics for LTL ^{$>$} with respect to [15], which is the main cause of the differences between our algorithm and previous ones. We will also discuss optimizations of the translation.

We label $\mathbf{R}_1^>, \dots, \mathbf{R}_k^>$ the k occurrences of the operator $\mathbf{R}^>$ in ϕ . Each occurrence is associated a counter, so that $\Gamma = \{\gamma_1, \dots, \gamma_k\}$. We note $\text{sub}(\phi)$ the set of sub-formulae of ϕ .

if $\psi = \psi_1 \wedge \psi_2$:	$\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\} \end{array} \right.$
if $\psi = \psi_1 \vee \psi_2$:	$\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\} \end{array} \right.$
if $\psi = \psi_1 \mathbf{U} \psi_2$:	$\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2\} \\ Y \xrightarrow[\mathbf{i}_\psi]{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \mathbf{X} \psi\} \end{array} \right.$
if $\psi = \psi_1 \mathbf{R} \psi_2$:	$\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2, \mathbf{X} \psi\} \end{array} \right.$
if $\psi = \psi_1 \mathbf{R}_i^> \psi_2$:	$\left\{ \begin{array}{l} Y \xrightarrow{\varepsilon:\mathbf{or}_i} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2\} \\ Y \xrightarrow[\varepsilon:\mathbf{i}_i]{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_1, \psi_2, \mathbf{X} \psi\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\psi_2, \mathbf{X} \psi\} \end{array} \right.$

Table 1: Reduction rules

the set of letters (in 2^{AP}) compatible with the literals in Z . Σ_Z cannot be empty if Z is consistent. From a reduced state $Z = \{l_1, \dots, l_n, \mathbf{X} \phi_1, \dots, \mathbf{X} \phi_p\}$, reading a letter of Σ_Z leads to the state $\text{next}(Z) = \{\phi_1, \dots, \phi_p\}$.

Non-reduced states are reduced step-by-step using ε -transitions, summarized in Table 1, that preserve the state semantics. Operators \vee, \wedge, \mathbf{U} and \mathbf{R} follow

A state of \mathcal{A}_ϕ is a set of LTL ^{$>$} formulae, yet to be verified. A formula is *reduced* if it is either a literal or its outermost operator is \mathbf{X} . A set Z of formulae is *reduced* if it contains only reduced formulae, and *consistent* if it does not contain both a formula and its negation. Given a reduced and consistent set Z , we note $\text{next}(Z) = \{\psi \mid \mathbf{X} \psi \in Z\}$ and Σ_Z

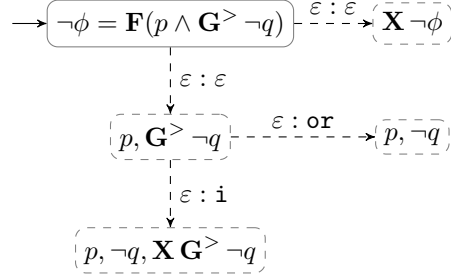


Figure 2: Reduction of $\{\neg\phi\}$

the classical translation from LTL to ω -automata. To reduce $\psi = \psi_1 \mathbf{R}_i^> \psi_2$, three ε -transitions are possible:

- the first one checks and resets the counter i , requiring both ψ_1 and ψ_2 to be verified;
- the second one counts one occurrence of ψ_1 , requiring all ψ_1 , ψ_2 and $\mathbf{X}\psi$ to be verified;
- the third one does nothing on the counter i , and requires both ψ_2 and $\mathbf{X}\psi$ to be verified.

These three transitions implement in fact the semantics of the operator $\mathbf{R}^>$.

An until formula $\phi_1 \mathbf{U} \phi_2$ requires ϕ_2 to be true at some point. Transitions subscripted with the label $!\psi$ indicate that ϕ_2 in ψ has been postponed. Each until sub-formula in ϕ yields one acceptance condition: any transition going through a label $!\psi$ is *not* accepting for the condition ψ . Once the automata with ε -transitions is built, the actual (smaller) automaton is built by collapsing ε -transitions (counter actions are concatenated).

Let us illustrate the described translation with an example: $\phi = \mathbf{G}(p \implies \mathbf{F}^{\leq} q)$. We turn ϕ into an equivalent LTL[>] formula $\neg\phi = \mathbf{F}(p \wedge \mathbf{G}^> \neg q)$, which we translate to a CA with sup-semantics. Figure 2 depicts the ε -transitions obtained while reducing $\{\neg\phi\}$.

The reduction yields three reduced sets. The (not reduced) set $\{p \wedge \mathbf{G}^> \neg q\}$ is not shown and is directly reduced to $\{p, \mathbf{G}^> \neg q\}$. The three reduced sets being also consistent, we are ready to find the real successors of $\neg\phi$, *i.e.* the sets $next(Z)$ where Z is one of the three obtained reduced sets. First, $next(\mathbf{X}\neg\phi)$ falls back to the initial state $\{\neg\phi\}$, which will result in a loop in the final automaton. Note that this will be the only non-accepting transition. $next(p, \neg q)$ is the set $\{\top\}$, and $next(p, \neg q, \mathbf{X} \mathbf{G}^> \neg q)$ is $\{\mathbf{G}^> \neg q\}$. As this last state is not reduced, the reduction process goes on, yielding $\{\neg q\}$ and $\{\mathbf{X} \mathbf{G}^> \neg q\}$. $next(\neg q) = \{\top\}$ and $next(\mathbf{X} \mathbf{G}^> \neg q) = \{\mathbf{G}^> \neg q\}$, states that have already been discovered and reduced. Finally, we collapse the ε -transitions to get the final automaton shown in Figure 3.

Several transitions of the automaton of Figure 3a are unnecessary. Indeed, according to the sup-semantics, only paths with the higher value are relevant, those with a lower value can safely be removed. This allows to reduce non-determinism in the automaton, as shown on the automaton of Figure 3b which has the same semantics as the one of Figure 3a.

If done appropriately, the actions in the produced automaton can be limited to atomic ones. The proof of Proposition 11 is detailed in appendix.

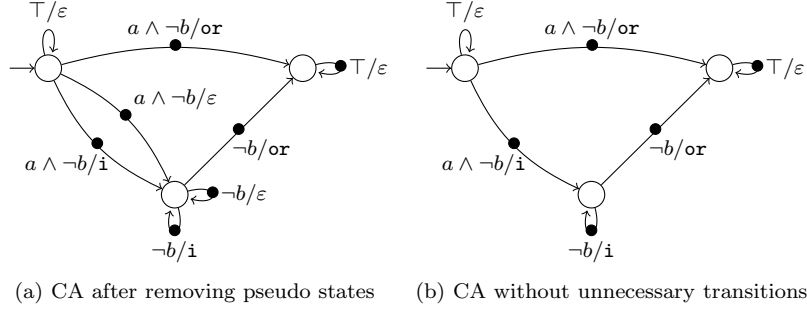


Figure 3: CA for $\mathbf{G}(p \implies \mathbf{F}^{\leq} q)$

Property 11. *If the largest (for the sub-formula ordering) formula in Y is picked first when reducing Y , then at most one action per counter occurs along any chain of ε -transitions.*

4.2 CEGAR-loop Implementation

We detail the implementation of `ComputeBound` using CA, specifically lines 4 and 5. The strength of our algorithm is to boil the problem down to ω -automata emptiness checks, a well-studied problem, with numerous variants and solutions [22, 23]. This section makes no further assumptions on the variant of ω -automata or the translation algorithm used, so that the final user can use the fittest ones. Many translations of LTL (and by extension CLTL translation derived from them) to automata produce generalized transition-based automata.

The input language L is assumed to be regular and given as an ω -automaton. As explained above, we build from ϕ a CA \mathcal{A}_ϕ such that $\llbracket \phi \rrbracket_{>} = \llbracket \mathcal{A}_\phi \rrbracket_{>}$. Line 4 looks for a word $u \in L$ such that $\llbracket \phi \rrbracket_{>}(u) > 0$. The constraint $u \in L$ is enforced by searching u such that $\llbracket \mathcal{A}_\phi \otimes L \rrbracket_{>}(u) > 0$, where $\mathcal{A}_\phi \otimes L$ is the synchronized product of \mathcal{A}_ϕ and (the automaton of) L . This product, itself a CA, rules out words not in L , so that $\llbracket \mathcal{A}_\phi \otimes L \rrbracket_{>}(v) > 0$ iff $v \in L$.

Proposition 8 shows that $\llbracket \phi \rrbracket_{>}(u) = 0$ iff there is no n such that $(u, n) \models \phi$. Given the requirements on \mathcal{A}_ϕ , $\llbracket \phi \rrbracket_{>}(u) = 0$ iff \mathcal{A}_ϕ has no accepting run on u . Thus, the set of such u 's is exactly the language recognized by $\mathcal{A}_\phi \otimes L$, viewed as a ω -automaton by ignoring the counters. Finding such a word u thus amounts to an emptiness check of the said automaton.

A non-empty regular ω -language contains an ultimately periodic word, and so can be chosen u , ensuring a finite representation. In practice, emptiness-check algorithms that compute a counter-example always produce such ultimately periodic words.

Line 5 then asks for a value p between n and $\llbracket \phi_0 \rrbracket_{>}(u)$ to update n . We claim that any accepting run ρ on u in the product automaton $\mathcal{A}_{\phi_0} \otimes \mathcal{A}_{\phi_0[n+1]}$ provides such a value p . On the one hand such a ρ is an accepting run in \mathcal{A}_{ϕ_0} , and its value p is therefore not larger than $\llbracket \phi_0 \rrbracket_{>}(u)$. On the other hand, ρ is also an accepting run in $\mathcal{A}_{\phi_0[n+1]}$. The whole point of synchronizing \mathcal{A}_{ϕ_0} with $\mathcal{A}_{\phi_0[n+1]}$ is to rule out runs of value strictly less than n . Indeed, in $\phi_0[n+1]$, the value $n+1$ is *hard-coded* thanks to $n+1$ nested \mathbf{X} operators. Every time a counter is incremented, a nested \mathbf{X} is passed, and $\mathcal{A}_{\phi_0[n+1]}$ accepts a run only

if counters are checked with values strictly larger than n . Therefore, replaying u in $\mathcal{A}_{\phi_0} \otimes \mathcal{A}_{\phi_0[n+1]}$ yields a p between n and $\llbracket \phi_0 \rrbracket_{>}(u)$. The great advantage of this operation is that only one run over u needs to be considered, and the computation of p is therefore straightforward.

We recall that higher p speed the convergence of **ComputeBound**, by reducing the number of loops. But higher p would require to explore several runs of $\mathcal{A}_{\phi_0} \otimes \mathcal{A}_{\phi_0[n+1]}$ and to retain the highest found value. We see more precisely here the trade-off between the number of loops in **ComputeBound** and the computation of p on line 5.

To conclude, we show how **ComputeBound** can be extended to also detect the unbounded case, thus providing a complete algorithm. To this end, we recall that unboundedness of a sup-automaton is decidable, as shown in [15].

Property 12. *[15] $\llbracket \mathcal{A} \rrbracket_{>}$ is unbounded if and only if \mathcal{A} has an accepting run ρ in which every action cr_γ ($\gamma \in \Gamma$) is preceded by a cycle that increments γ without resetting γ .*

The proof of Proposition 12 is not difficult: the existence of such a cycle guarantees the ability to build runs with arbitrarily high values. Conversely, if no such run exists, then every accepting run has its value bounded by $|Q_{\mathcal{A}}|$.

As a corollary of Proposition 12, $\sup \llbracket \mathcal{A} \rrbracket_{>}$ is unbounded if and only if \mathcal{A} has an accepting run of value greater than its number of states. **ComputeBound** can thus be adapted so as to detect unboundedness too: compute a bound B on the size of the product $\mathcal{A}_{\phi_0} \otimes \mathcal{A}_L$ (such as $B = |\mathcal{A}_{\phi_0}| \times |\mathcal{A}_L|$). The sought bound is finite iff n ever exceeds B .

5 An example of Application

Through a concrete example, this section illustrates the expressive capabilities of LTL^{\leq} and $\text{LTL}^>$ and the kind of problems our bound evaluation algorithm may solve.

5.1 Ant Colony Optimization

Ant Colony Optimization [9] (ACO) is a bio-inspired meta-heuristic relying on the cooperative behavior of small simple agents to solve optimization problems. A collection of artificial *ants* endlessly walk a graph randomly, from some initial node (their *nest*), to one or several target nodes (the *sources of food*), and come back to the nest. Whenever an ant moves from one node to another, it deposits a certain amount of pheromone. The quantity of pheromone left on an edge increases the likelihood that an ant chooses to cross it. Besides, the quantity of pheromone decreases according to an *evaporation rate*. Unless the evaporation rate is too high, ants will eventually converge to the shortest paths from their nest to the food sources, because shorter paths will be rewarded with new pheromone more frequently.

ACO has been successfully used in numerous applications, such as data mining [21], image processing [20]. ACO is resilient to modifications of the graph and it usually responds very quickly to such changes because its current state is likely to contain useful information on the closest new solutions. Finally, ACO is rather simple to implement on huge distributed setups, as agents do not communicate directly with each others.

5.2 Quantitative Properties

Let us consider an ACO that searches the shortest path between two nodes in a directed graph. The most classic quantitative information is the time (number of steps) taken to find a solution, be it local or global. Topological parameters may also be measured, such as the maximum length of solutions or the number of nodes visited before a solution is found. Other quantities relate to the algorithm parameters, such as the maximum amount of pheromones on an edge. Such information is critical to tune algorithm parameters, that ultimately dictate how fast it converges to a solution [12]. With a fixed topology, some of these properties are not difficult to compute. For instance, the minimum number of visited nodes is the length of the shortest path, computable in polynomial time. Other properties are harder to compute, such as the maximum length of solutions. When the topology dynamically changes, analytical search for exact optima values is cumbersome, if even possible.

To address these questions, model-checking becomes an option, by checking all possible behaviors of the system. A common approach instruments the model to monitor the quantitative properties at stake. It introduces a strong semantical risk, because instrumentation may be impacted by any modification to the model, and must thus be kept up-to-date. We propose to move the instrumentation into the logics, to keep a proper separation between the actual behavior (the model) and the desired behavior (the logical property).

Let $G = \langle V, E \rangle$ be a directed graph where V is a finite set of vertices and $E \subseteq V \times V$ is the set of edges. An ant is a pair $\langle a_v, a_d \rangle$ where $a_v \in V$ is a node, $a_d \in \{\uparrow, \downarrow\}$ is a direction (looking for a food source, and coming back to the nest). A denotes the set of ants.

The time an ant a takes to find a solution is given by $\phi_a(a) = \mathbf{G}(\perp \mathbf{U}^{\leq} (a_v = s \wedge a_d = \downarrow))$ where $s \in V$ is the nest node. The worst-case over possible behaviors in L is thus $\sup_L \llbracket \phi_a \rrbracket_{\leq}$ and the best-case $\inf_L \llbracket \phi_a \rrbracket_{\leq}$. Similarly, the time taken by the *whole system* to find a solution is obtained by the conjunction over all ants of the previous: $\phi_A = \mathbf{G} \bigwedge_{a \in A} \phi_a(a)$.

It is easy to count events like the number of visits of an ant a to a node $s \in V$ with $(a_v = s) \mathbf{U}^{\leq} (\mathbf{G} \neg(a_v = s))$. Occurrences of a position where a LTL formula ϕ holds are counted by $\phi \mathbf{U}^{\leq} (\mathbf{G} \neg \phi)$. Consider the deposit (resp. removal) of a pheromone on edge e , denoted by action $\mathbf{add}(e)$ (resp. $\mathbf{rm}(e)$). The formula $\phi_{\text{acc}}(e) \equiv \neg(\mathbf{add}(e) \implies (\neg \mathbf{add}(e) \mathbf{U} \mathbf{rm}(e)))$ holds in states where e will receive more pheromone before the next removal, i.e. when an ant crosses an edge whose pheromones have not yet evaporated. The (integer) amount of pheromone on a given edge e is obtained by $\phi_{\text{acc}}(e) \mathbf{U}^{\leq} (\mathbf{G} \neg \phi_{\text{acc}}(e))$.

6 Related work

A famous problem in language theory is the star-height problem: given a language L (of finite words) and an integer k , is there a regular expression for L with at most k nested Kleene stars? Proposed in 1963 [11], it was proven decidable in 1988 [13] by exhibiting an algorithm with non-elementary complexity, and a much more efficient algorithm was then proposed in 2005 [14]. Both algorithms translate the problem to the existence of a bound for a function mapping words to integers, represented in both cases by an automaton equipped with coun-

ters (distance automata for the former, nested distance desert automata for the latter). This boundedness problem of the existence of a bound is then shown decidable. It is the first of many problems that reduce to the boundedness problem for such automata.

This motivated an in-depth study of automata with counters (as we use it) as a general framework, that came up with a theory extending the one of regular languages, with logical and algebraic counter-parts [6]. On infinite words, the logical counter-part motivated the introduction and study of LTL^{\leq} and $LTL^{>}$ [16]. This theory also encompasses *promptness* properties, a variant of liveness where a bound on the wait time of a recurring event must exist [2, 17]. But all these works, motivated by the boundedness problem, overlook the exact values of the functions. On one hand, this relaxation enables nice closure properties (such as the equivalent expressiveness for inf-automata and sup-automata). On the other hand, it only allows to reason about the existence of a bound, not to compute values.

In verification, not all questions have a boolean answer, so that various quantitative extensions of automata have been considered, such as weighted automata (see [10] for a survey). Despite their various domains of application, they have limited expressivity, as the domain of weights is required to be a semi-ring. An extension to arbitrary operations on weights have been recently proposed [3]. It encompasses various extensions of weighted automata, such as Discounted Sum Automata [7] and Counter ω -Automata as considered in this paper. All these formalisms can be characterized by the absence of guards on register values. These extensions sometimes have equivalent logics (such as discounted linear temporal logics [1]). From the logical point of view, let us also mention that other temporal logics able to count events were previously proposed [19].

Most of the cited works only focus on expressivity, decidability and complexity problems, with little consideration to the practical use of such quantitative extensions of automata. It contrasts with older formalisms: ω -automata have already received great focus towards practical applications, illustrated by numerous emptiness checks algorithms (see [22] for an overview) and many implementations, principally oriented towards LTL model-checking (see [23] for a survey). Some quantitative extensions of automata possess a similar maturity towards practical applications, especially timed automata [4] and weighted automata [18].

7 Conclusion

In this paper we proposed to use LTL^{\leq} and $LTL^{>}$ for practical verification of quantitative properties. One key advantage of these logics is to clearly separate functional properties of the system and quantitative properties, expressed in the logic used for verification. The functional model can still be used for other tasks like production of code and test generation. Along with examples of properties to be expressed with these logics, we also exhibit a CEGAR-like algorithm to compute bounds for such formulae, based on successive refinements. We further proposed an implementation of this algorithm using automata equipped with counters, extending the automata approach used for LTL model-checking.

This is a first step towards practical applications of such logics which seems

very promising if adequate algorithms and tools are available. The next step is to implement our algorithm in a proof-of-concept tool. The logics we used are just a drop in a vast ocean of quantitative extensions for LTL. Further research should focus on fitting our algorithm in a more general framework so as to capture several such LTL extensions. Another axis would be the improvement of the performance of validation algorithms: such as improving the translation to automata to produce smaller and/or more deterministic automata and tweak emptiness checks to limit the number of refinement iterations.

References

- [1] S. Almagor, U. Boker, and O. Kupferman. Discounting in LTL. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCIS*, pages 424–439. Springer Berlin Heidelberg, 2014.
- [2] S. Almagor, Y. Hirshfeld, and O. Kupferman. Promptness in ω -regular automata. In *Proc. 8th International Symposium on Automated Technology for Verification and Analysis (ATVA '10)*, volume 6252 of *LNCIS*, pages 22–36. Springer, 2010.
- [3] R. Alur, L. Dantoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular Functions and Cost Register Automata. In *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on*, pages 13–22. IEEE, 2013.
- [4] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Proc. 3rd International Conference on the Quantitative Evaluation of Systems, QEST '06*, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society.
- [5] M. Bojańczyk and T. Colcombet. Bounds in ω -regularity. In *Proc. 21st Annual IEEE Symposium on Logic in Computer Science, LICS '06*, pages 285–296, Washington, DC, USA, 2006. IEEE Computer Society.
- [6] T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming*, pages 139–150. Springer, 2009.
- [7] L. de Alfaro, T. Henzinger, and R. Majumdar. Discounting the Future in Systems Theory. In JosC.M. Baeten, J. Lenstra, J. Parrow, and G.. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer Berlin Heidelberg, 2003.
- [8] S. Demri and P. Gastin. Specification and verification using temporal logics. In D. D’Souza and P. Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*, chapter 15, pages 457–494. World Scientific, July 2012.
- [9] M. Dorigo and L. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1997.

- [10] M. Droste and P. Gastin. Weighted Automata and Weighted Logics. *Theoretical Computer Science*, 380(1):69–86, 2007.
- [11] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10(4):385–397, 12 1963.
- [12] D. Gaertner and K. Clark. On optimal parameters for ant colony optimization algorithms. In *Proc. International Conference on Artificial Intelligence 2005*, pages 83–89. CSREA Press, 2005.
- [13] K. Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation*, 78(2):124 – 169, 1988.
- [14] D. Kirsten. Distance desert automata and the star height problem. *RAIRO-Theoretical Informatics and Applications*, 39(03):455–509, 2005.
- [15] D. Kuperberg. Linear temporal logic for regular cost functions. *Logical Methods in Computer Science*, 10(1), 2014.
- [16] D. Kuperberg and M. Vanden Boom. On the expressive power of cost logics over infinite words. In *Automata, Languages, and Programming*, pages 287–298. Springer, 2012.
- [17] O. Kupferman, N. Piterman, and M. Vardi. From liveness to promptness. In *Computer Aided Verification*, pages 406–419. Springer, 2007.
- [18] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV’11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- [19] F. Laroussinie, A. Meyer, and E. Pettonnet. Counting LTL. In *Proc. 2010 17th International Symposium on Temporal Representation and Reasoning, TIME ’10*, pages 51–58, Washington, DC, USA, 2010. IEEE Computer Society.
- [20] H. Nezamabadi-pour, S. Saryazdi, and E. Rashedi. Edge detection using ant algorithms. *Soft Computing*, 10(7):623–628, 2006.
- [21] R.. Parpinelli, H. Lopes, and A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6:321–332, 2002.
- [22] E. Renault, A. Duret-Lutz, F. Kordon, and D. Poitrenaud. Three SCC-based Emptiness Checks for Generalized Büchi Automata. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proc. 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR’13)*, volume 8312 of *LNCS*, pages 668–682. Springer, 2013.
- [23] K. Rozier and M. Vardi. LTL Satisfiability Checking. *International journal on software tools for technology transfer*, 12(2):123–137, 2010.

A Proof of Property 1

Proof. The proof proceeds by structural induction on ϕ . Note that $\phi[n] = \phi$ for any n if $\phi \in \text{LTL}$, therefore the property holds on the LTL fragment (and in particular on literals).

We recall that $\llbracket \phi \rrbracket_{\leq}(u) \leq n$ iff $(u, n) \models_{\leq} \phi$.

Let us assume that the property holds for LTL^{\leq} formulae ϕ_1 and ϕ_2 . Let $\bowtie \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}$. $(\phi_1 \bowtie \phi_2)[n] \vdash u$ iff $\phi_1[n] \bowtie \phi_2[n] \vdash u$. The induction hypothesis allows to replace every occurrence of $\phi_i \vdash v$ in the usual LTL semantics of \bowtie by $(v, n) \models_{\leq} \phi_i$. This gives the LTL^{\leq} semantics of \bowtie , thus proving that if $(\phi_1 \bowtie \phi_2)[n] \vdash u$, then $(u, n) \models_{\leq} \phi_1 \bowtie \phi_2$. The converse reasoning (from LTL^{\leq} to LTL semantics) proves the converse implication. The same argument is applied to the case $\phi = \mathbf{X} \phi_1$.

Now consider $\phi = \phi_1 \mathbf{U}^{\leq} \phi_2$. In the general case, $\phi[n] = (\phi_1[n] \mathbf{U}^{\leq} \phi_2[n])[n]$. Suppose we have a proof for this case when ϕ_1 and ϕ_2 are LTL formulae. The induction hypothesis allows to replace any occurrence of ϕ_1 and ϕ_2 in such a proof by $\phi_1[n]$ and $\phi_2[n]$, using the same argument as presented above. Thus, it suffices to prove the property when ϕ_1 and ϕ_2 are LTL formulae to conclude the proof.

Assume ϕ_1 and ϕ_2 are LTL formulae. We now proceed by induction on n . If $n = 0$, then $\phi[n] = \phi_1 \mathbf{U} \phi_2$. $u \vdash \phi[n]$ if, and only if, for some index i , $u^i \vdash \phi_2$ and $u^j \models_{\leq} \phi_1$ for every $j < i$. In other words, $u \vdash \phi[n]$ if, and only if, for some index i , $u^i \vdash \phi_2$ and $\{j < i \mid u^j \not\models_{\leq} \phi_1\} = \emptyset$. Thus, $u \vdash \phi[n]$ if, and only, if $(u, 0) \models_{\leq} \phi$.

If the property holds at n , then $\phi[n+1] = (\phi_1 \vee \mathbf{X}(\phi[n])) \mathbf{U} \phi_2 = (\phi_1 \mathbf{U} \phi_2) \vee (\mathbf{X}(\phi[n]) \mathbf{U} \phi_2)$. If $u \vdash (\phi_1 \mathbf{U} \phi_2)$, then $(u, 0) \models_{\leq} \phi$ as above, or equivalently $\llbracket \phi \rrbracket_{\leq}(u) \leq 0 < n+1$. If $u \vdash \mathbf{X}(\phi[n]) \mathbf{U} \phi_2$, then there is some index i such that $u^i \vdash \phi_2$ and $u^{j+1} \vdash \phi[n]$ for every $j < i$. Again by induction hypothesis, $(u^i, n+1) \models_{\leq} \phi_2$. Let us now consider $S = \{j < i \mid (u^j, n+1) \not\models_{\leq} \phi_1\}$. Let $j < i$. We know that $u^{j+1} \vdash \phi[n]$ which is equivalent, by induction hypothesis, to $(u^{j+1}, n) \models_{\leq} \phi$. Therefore, there exists an index $i_j > j$ such that $(u^{i_j}, n) \models_{\leq} \phi_2$. Since $(u^i, n+1) \models_{\leq} \phi_2$, we necessarily have $i_j \geq i$. S is a subset of $T = \{j < i \mid (u^j, n) \not\models_{\leq} \phi_1\}$. We know that $T - \{0\}$ is of size at most n (in case $(u^k, n) \models_{\leq} \phi$ for every $i \leq k < i_0$). Therefore, S is of size at most $n+1$, which concludes the proof. \square

B Proof of Property 11

Proof. Consider a path of ε -transitions from a non-reduced state Y' to a reduced state Y . Whenever a formula ψ is reduced along this path, it is removed from the current state ($\mathbf{X} \psi$ may appear, but it cannot be reduced until after Y'), and only strict sub-formulae of ψ are added. We claim that the operator $\mathbf{R}_i^>$ (that occurs only once in ψ) is reduced at most once along the path (for every i). Indeed, when $\psi = \psi_1 \mathbf{R}_i^> \psi_2$ is reduced once, the only way to have it reduced a second time is to be added to the current set by the reduction of another formula ψ' . It implies that ψ is a sub-formula of ψ' . Since ψ cannot be a sub-formula of one of its strict sub-formulae, then there was a non-reduced formula ψ'' with ψ as strict sub-formula when ψ was reduced, which contradicts the selection procedure of formulae to be reduced. \square